(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
31 January 2002 (31.01.2002)

PCT

(10) International Publication Number
WO 02/08859 A2

(51) International Patent Classification[7]: G06F

(21) International Application Number: PCT/US01/23084

(22) International Filing Date: 20 July 2001 (20.07.2001)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
09/624,437 21 July 2000 (21.07.2000) US

(71) Applicant: **CHICORY SYSTEMS, INC.** [US/US]; 12365 Riata Terrace Parkway, Building II, Suite 150, Austin, TX 78727 (US).

(72) Inventors: **DERRICK, John, E.**; 2424 Falcon Drive, Round Rock, TX 78681 (US). **MCDONALD, Robert, G.**; 6280 McNeil Drive, Apt. 507, Austin, TX 78729 (US).

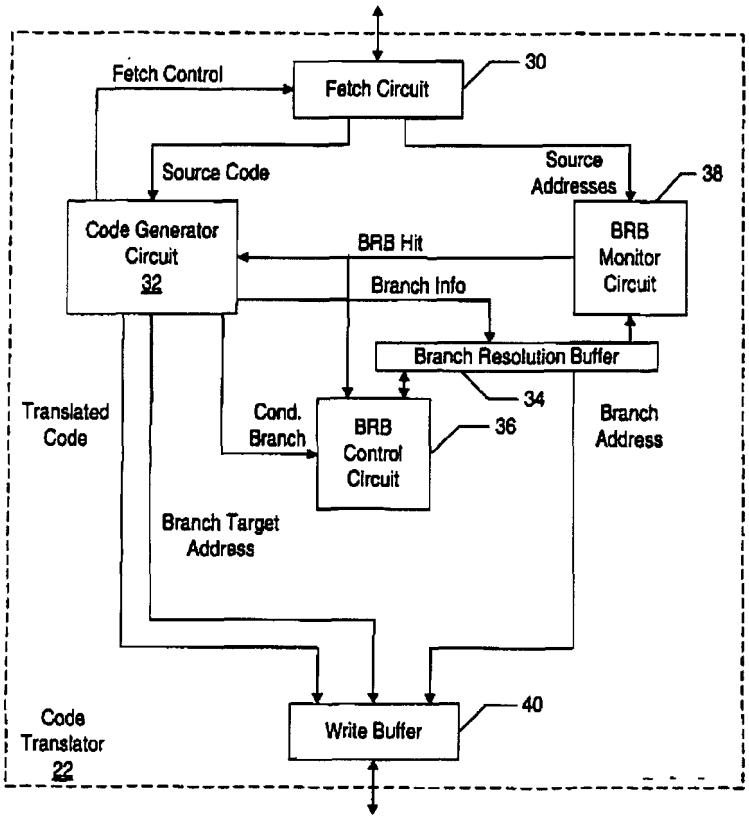
(74) Agent: **MERKEL, Lawrence, J.**; Conley, Rose & Tayon, P.C., Patent Agent, P.O. Box 398, Austin, TX 78767-0398 (US).

(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE, TR), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: CODE TRANSLATOR CAPABLE OF TRANSLATING MULTIPLE CONTROL PATHS



(57) Abstract: A code translator may translate instructions from a source code sequence to a translated code sequence. If a conditional branch instruction is detected, the code translator may translate instructions from both the sequential path and the target path of the conditional branch instruction. The translated conditional branch instruction may be coded to generate a branch target address identifying the first instruction in the target path within the translated code sequence. In one embodiment, the code translator selects one of the sequential path and the target path on which to continue translation after encountering the conditional branch instruction. The address of the other path and the address of the translated conditional branch instruction may be recorded in a buffer. If the code translator detects a terminating condition in the selected path, instructions from the other path are translated and the translated conditional branch instruction is coded to generate a branch target address identifying the first translated instruction from the other path in the translated code sequence.

WO 02/08859 A2

Published:
— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

TITLE: CODE TRANSLATOR CAPABLE OF TRANSLATING MULTIPLE CONTROL PATHS

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention is related to the field of programmable computing systems and, more particularly, to handling branch instructions while translating between instruction sets in computing systems.

2. Description of the Related Art

A variety of instruction sets have been developed over time for use in central processing units (CPUs). Each instruction set defines the instructions supported by a CPU implementing the instruction set and the binary encodings of those instructions. Each instruction in the instruction set is assigned a unique encoding which distinguishes that instruction from the other instructions in the instruction set.

Generally, program code is written using the instructions from one instruction set. Unfortunately, the only CPUs which may directly execute the code are CPUs which implement that instruction set. To be executed on other CPUs, the instructions forming the code may be emulated using additional software which interprets the instructions in the program code and performs equivalent instruction sequences using the native instruction set of the CPU. The resulting program execution performance may be substantially lower, and frequently may be unacceptably slow.

In some cases (such as the Java language), software just-in-time (JIT) compilers have been proposed. A Java JIT compiler complies Java bytecodes into instructions specified by the native instruction set of the CPU upon which execution is desired. While executing the compiled code is faster than emulated execution, the software compilation process itself is relatively time consuming. Thus, a large amount of memory is typically dedicated to storing the compiled code, so that the amount of time required to perform the compilation may be absorbed by performing the compilation once and allowing for the compiled code to be executed many times.

While the JIT compiler provides for speedier execution, the large amount of memory required to store the compiled code makes the JIT compiler unsuitable for certain types of machines. For example, set top boxes, personal digital assistants, and other hand-held computing devices generally have a limited amount of memory. Thus, dedicating a large amount of memory to store compiled code is not possible in these types of computing devices.

Furthermore, the handling of conditional branch instructions is complex in the various solutions for executing non-native code on a CPU. Since the next instructions to be executed after a conditional branch instruction may be from one of two paths (the sequential path and the target path), which instructions to translate after a conditional branch instruction is unknown.

SUMMARY OF THE INVENTION

The problems outlined above are in large part solved by a system including a code translator as described herein. The code translator may translate instructions from a source code sequence coded using instructions defined in a first instruction set to a translated code sequence coded using instructions defined in a second

instruction set. If a conditional branch instruction is detected, the code translator may translate instructions from both the sequential path and the target path of the conditional branch instruction. The translated conditional branch instruction may be coded to generate a branch target address identifying the first instruction in the target path within the translated code sequence. In this manner, the conditional branch instruction from the source sequence may be efficiently handled within the translated sequence.

In one embodiment, the code translator selects one of the sequential path and the target path on which to continue translation after encountering the conditional branch instruction. The address of the other path (within the source code sequence) and the address of the translated conditional branch instruction (within the translated code sequence) may be recorded in a buffer. If the code translator detects a terminating condition in the selected path, instructions from the other path are translated and the translated conditional branch instruction is coded to generate a branch target address identifying the first translated instruction from the other path in the translated code sequence. For a forward branch, a terminating condition may be detecting the merging of the sequential and target paths (by monitoring addresses of instructions to be translated against the addresses of the other paths, as recorded in the buffer). Other termination conditions may include, for example, an untranslatable instruction in the selected path.

Broadly speaking, an apparatus is contemplated, comprising a first circuit and a buffer. The first circuit is configured to translate a first conditional branch instruction in a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions in a second instruction set. The first circuit is configured to continue translation down a first path of a target path or a sequential path of the first conditional branch instruction. Additionally, the first circuit is configured to output a first address of the second conditional branch instruction within the translated code sequence and a second address of a first instruction within a second path of the target path or the sequential path. The buffer includes at least a first entry configured to store the first address and the second address. The first circuit is configured to translate at least the first instruction located by the second address within the second path to a second instruction in the translated code sequence responsive to a terminating condition in the first path. Furthermore, the first circuit is configured to code the second conditional branch instruction at the first address to generate a branch target address, during execution, which identifies the second instruction in the translated code sequence.

Additionally, a method is contemplated. A first conditional branch instruction is translated from a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions defined in a second instruction set. A first address locating the second conditional branch instruction in the translated code sequence is recorded. Also, a second address locating a first instruction of a first path of a target path or a sequential path of the first conditional branch instruction is recorded. Translation continues down a second path of the target path or the sequential path. A termination condition is detected in the second path. At least the first instruction of the first path is translated to a second instruction in the translated code sequence. The first instruction is located at the second address. The second conditional branch instruction is coded to generate a branch target address, during execution, which identifies the second instruction. The second conditional branch instruction is located at the first address.

Furthermore, a system is contemplated, comprising a central processing unit (CPU) and a code translated coupled to the CPU. The CPU is configured to execute instructions defined in a first instruction set. The code

translator is configured to translate a source code sequence coded with instructions defined in a second instruction set to a translated code sequence coded within instructions defined in the first instruction set. The code translator includes a buffer and is configured, responsive to translating a first conditional branch instruction in the source code sequence to a second conditional branch instruction in the translated code sequence, to continue translation down a first path of a sequential path or a target path of the first conditional branch instruction and to record a first address of the second conditional branch instruction within the translated code sequence and a second address of a first instruction of a second path of the sequential path or the target path in the buffer. Additionally, the code translator is configured to detect a terminating condition in the first path and to translate at least the first instruction within the second path located at the second address to a second instruction in the translated code sequence responsive to detecting the terminating condition. Still further, the code translator is configured to code the second conditional branch instruction to generate a branch target address, during execution, which identifies the second instruction in the translated code sequence. The second conditional branch instruction is located at the first address. Still further, a carrier medium configured to carry a database is contemplated. The database is representative of the code translator.

Moreover, an apparatus is contemplated, comprising a first circuit configured to translate a first conditional branch instruction in a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions in a second instruction set. The first circuit is configured to translate one or more instructions from a first path of a target path or a sequential path of the first conditional branch instruction, and at least a first instruction within a second path of the target path or the sequential path to a second instruction in the translated code sequence. The first circuit is configured to code the second conditional branch instruction to generate a branch target address, during execution, which identifies the second instruction in the translated code sequence.

Additionally, a method is contemplated. A first conditional branch instruction is translated from a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions defined in a second instruction set. A first path of a target path or a sequential path of the first conditional branch instruction is translated down. At least a first instruction of a second path of the target path or the sequential path of the first conditional branch instruction is translated to a second instruction in the translated code sequence. The second conditional branch instruction is coded to generate a branch target address, during execution, which identifies the second instruction.

BRIEF DESCRIPTION OF THE DRAWINGS

Other objects and advantages of the invention will become apparent upon reading the following detailed description and upon reference to the accompanying drawings in which:

Fig. 1 is a block diagram of one embodiment of a system including a code translator.

Fig. 1A is a block diagram of another embodiment of a system including a code translator.

Fig. 2 is a block diagram of one embodiment of the code translator shown in Figs. 1 and 1A, including a branch resolution buffer.

Fig. 3 is a block diagram of one embodiment of a branch resolution buffer entry.

Fig. 4 is a flowchart illustrating operation of one embodiment of the code translator for translating an

instruction.

Fig. 5 is a flowchart illustrating operation of one embodiment of the code translator for detecting the beginning of translation of a second path of a branch instruction.

Fig. 6 is an exemplary source and translated code sequence.

Fig. 7 is a second exemplary source and translated code sequence.

Fig. 8 is a flowchart illustrating an exemplary method for allocating branch resolution buffer entries.

Fig. 9 is a block diagram of a carrier medium.

While the invention is susceptible to various modifications and alternative forms, specific embodiments thereof are shown by way of example in the drawings and will herein be described in detail. It should be understood, however, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Turning now to Fig. 1, a block diagram of one embodiment of a system 10 is shown. Other embodiments are possible and contemplated. The illustrated system 10 includes a central processing unit (CPU) 12, a memory controller 14, a memory 16, a Peripheral Component Interconnect (PCI) bridge 18, a PCI bus 20, and a code translator 22. CPU 12 is coupled to PCI bridge 18 and memory controller 14. Memory controller 14 is further coupled to memory 16. PCI bridge 18 is further coupled to PCI bus 20. Code translator 22 is coupled to PCI bus 20. In one embodiment, CPU 12, memory controller 14, and PCI bridge 18 may be integrated onto a single chip or into a package as illustrated by the dotted line surrounding these components in Fig. 1 (although other embodiments may provide these components separately).

Generally, CPU 12 is capable of executing instructions defined in a first instruction set (the native instruction set of system 10). The native instruction set may be any instruction set, e.g. the ARM instruction set, the PowerPC instruction set, the x86 instruction set, the Alpha instruction set, etc. Code translator 22 is provided for translating code sequences coded using a second instruction set, different from the native instruction set, to a code sequence coded using the native instruction set. Code sequences coded using the second instruction set are referred to as "non-native" code sequences, and code sequences coded using the first instruction set of CPU 12 are referred to as "native" code sequences.

When CPU 12 detects that a non-native code sequence is to be executed, CPU 12 may communicate the source address of the beginning of the non-native code sequence and the destination address at which code translator 22 is to write the translated code sequence to code translator 22. Code translator 22 reads the non-native code sequence from the source address, translates the non-native code sequence to a native code sequence, and stores the native code sequence at the destination address. Code translator 22 may include a status register storing the status of the translation. Code translator 22 may signal CPU 12 that the translation is complete (e.g. through an interrupt or a coprocessor done signal), or CPU 12 may periodically poll code translator 22 to determine when the translation is complete (e.g. by reading the status register). Once the translation is complete, CPU 12 may read the status register to ensure that no errors occurred during the translation, and may then execute the native code sequence stored at the destination address. It is noted that the source and destination addresses are addresses

identifying memory locations within the memory 16. As an alternative to receiving the destination address from CPU 12, code translator 22 may maintain a cache of translated code sequences (internally or in a designated area of memory 16) and may provide the destination address of the routine upon completion of the translation.

In one embodiment, code translator 22 is configured to translate Java code sequences to the native instruction set. Thus, Java bytecodes may be used as an example of a non-native instruction set below. However, the techniques described below may be used with any non-native instruction set. Additionally, the Java instruction set uses a stack-based programming and storage model, while the native instruction set may use a register-based programming and storage model. The techniques described below for converting between the Java instruction set and the register-based native instruction set are applicable to converting any other stack-based instruction set. As used herein, the terms "register-based programming and storage model" or "register-based instruction set" refer to a model or instruction set in which operands for instructions are stored in a set of registers defined by the architecture. Each register is identified via a register index, and the register indexes are coded into the instructions to specify the operands of the instructions. Operand fetch for instructions in a register-based instruction set are then generally reads of the registers, typically implemented within the CPU. Register-based instruction sets often use explicit load/store instructions to load operands from memory locations to registers for subsequent instructions to use as operands and to store results from registers to memory locations. Furthermore, the term "instruction set" as used herein refers to a group of instructions defined by a particular architecture. Each instruction in the instruction set may be assigned an opcode which differentiates the instruction from other instructions in the instruction set, and the operands and behavior of the instruction are defined by the instruction set. Thus, Java bytecodes are instructions within the instruction set specified by the Java language specification, and the term bytecode and instruction will be used interchangeably herein when discussing Java bytecodes. Similarly, ARM instructions are instructions specified in the ARM instruction set, PowerPC instructions are instructions specified in the PowerPC instruction set, etc.

Since code translator 22 may translate from a stack-based instruction set to a register-based instruction set, code translator 22 may include hardware for translating the stack references in the stack-based instruction set to register indexes in the register-based instruction set. More particularly, a subset (or "pool") of the registers may be reserved to store stack operands. Code translator 22 may assign register indexes as stack items are pushed onto the stack, and may use those register indexes as source operands for instructions which reference the stack. After the stack items are popped from the stack, the corresponding registers may be free for use for another stack item pushed onto the stack. Thus, the register pool may store the topmost items on the stack, and memory may be used for lower items. The register-based instruction set may be most efficient at accessing operands in registers (since loads and stores may be needed to read the values from memory), and thus keeping items at the top of the stack in registers may enhance performance.

As an alternative to reserving the pool of registers, code translator 22 may be configured to statically or dynamically allocate registers from the register set of CPU 12 into the register pool. Code translator 22 may generate native instructions to store the registers selected for the pool to a scratchpad memory area (preserving the values in the selected registers), and then these registers may be used to store stack items. In a static embodiment, the entire pool of registers may be allocated at the beginning of a translated code sequence. In a dynamic embodiment, registers may be allocated to the pool as additional registers are needed during the translation. At the

end of the translated code sequence, code translator 22 may insert instructions to restore the values of these registers by reading the values from the scratchpad memory area (after storing the items to the operand stack).

In one embodiment, code translator 22 may translate instructions beginning at the source address and until a terminating condition in the source code sequence is reached. For example, a terminating condition may be a non-native instruction which code translator 22 is not configured to translate (e.g. because the instruction is too complex to translate efficiently). The non-native instruction may be emulated instead. As another example, a terminating condition may be a maximum number of instructions translated. The maximum number may be the number of source instructions (e.g. non-native instructions) or the number of translated instructions (e.g. native instructions). Alternatively, the number of bytes may be limited (and may be either the number of bytes of source instructions or the number or bytes of translated instructions). The maximum number of bytes/instructions may be programmable in a configuration register of code translator 22 (not shown). In one particular implementation, for example, a maximum size of 64 or 128 bytes of translated code may be programmably selected. As described in more detail below, a terminating condition when translating the sequential path of a forward branch may be the detection of the merge of the sequential path with the target path.

Code translator 22 may attempt to handle branch instructions efficiently in translating code sequences. Unconditional branch instructions (which always branch to the branch target address) may be deleted (or "folded out") of the translated code sequence. Instructions at the branch target address in the source code sequence may be inserted in the translated code sequence consecutive to instructions prior to the unconditional branch instruction. On the other hand, conditional branch instructions may cause instruction execution to continue with the sequential path or the target path, based on the results of some preceding instruction. Upon encountering a conditional branch instruction in the source code sequence, code translator 22 may generate a translated branch instruction and may continue translation down one of the sequential path or target path of the conditional branch instruction within the source code sequence. The code translator may record the address of the other path within the source code sequence and the address of the translated branch instruction in the translated code sequence which corresponds to the branch instruction in the source code sequence. Upon reaching a terminating condition in the selected path, the code translator may translate one or more instructions from the other path. The translated instructions corresponding to the one or more instructions from the other path are inserted into the translated code sequence. Additionally, the code translator may code the translated branch instruction to generate a branch target address identifying the first instruction of the translated instructions corresponding to the other path (i.e. the branch target address is the address of the first instruction of the translated instructions). In this manner, a translated code sequence including instructions from both the target path and the sequential path of the conditional branch instruction may be generated. The branch instruction may be handled efficiently within the code sequence, rather than returning to a control program executed by CPU 12 if the conditional branch instruction selects an untranslated path during execution. More details regarding one embodiment of the handling of branch instructions in code translator 22 are provided further below.

Generally, CPU 12 executes native code sequences and controls other portions of the system in response to the native code sequences. More particularly, CPU 12 may execute a control program which is used to communicate with code translator 22 to control translation of code sequences. Code translator 22 may terminate each translated code sequence with an exit instruction which returns control to the control program. More

particularly, the exit instruction may be an unconditional branch having a predefined target address within the control program. Conditional exit instructions may also be used, as described in more detail below. A conditional exit instruction may be a conditional branch having a predefined target address within the control program. The predefined target address may be a routine which determines if an untranslated instruction or other exception condition has been encountered (and may handle the exception condition) and may further determine the next code sequence to be executed (if already translated and cached in memory 16) or translated. The control program may handle untranslated instructions and other exception conditions with respect to the non-native code. In one embodiment in which the non-native instruction set is the Java instruction set, the control program may be part of the JVM for system 10. The JVM may include the interpreter mode to handle untranslated instructions and exception conditions detected by code translator 22. The JVM executed by CPU 12 may include all of the standard features of a JVM and may further include code to activate code translator 22 when a Java code sequence is to be executed, and to jump to the translated code after code translator 22 completes the translation. Code translator 22 may insert a return instruction to the JVM at the end of each translated sequence. CPU 12 may further execute the operating system code for system 10, as well as any native application code that may be included in system 10.

Memory controller 14 receives memory read and write operations from CPU 12 and PCI bridge 18 and performs these read and write operations to memory 16. It is noted that some of the read and write operations presented by PCI bridge 18 may be read and write operations generated by code translator 22 (e.g. read operations from the source address and subsequent addresses and write operations to the destination address and subsequent addresses). Memory 16 may comprise any suitable type of memory, including SRAM, DRAM, SDRAM, RDRAM, or any other type of memory.

PCI bridge 18 facilitates communication between PCI bus 20 and memory controller 14 or CPU 12. More particularly, PCI bridge 18 may detect read or write operations to the addresses which map to code translator 22, and may transmit those operations on PCI bus 20 to code translator 22. PCI bridge 18 may also detect read and write operations from code translator 22 to memory 16 on PCI bus 20 and may transmit those operations to memory controller 14.

It is noted that, while the PCI bus is used as an exemplary peripheral bus in the embodiment of Fig. 1, any other bus may be used. For example, the Universal Serial Bus (USB), IEEE 1394 bus, the Industry Standard Architecture (ISA) or Enhanced ISA (EISA) bus, the Personal Computer Memory Card International Association (PCMCIA) bus, etc. may be used. Still further, the Advanced RISC Machines (ARM) Advanced Microcontroller Bus Architecture (AMBA) bus, including the Advanced High-Performance (AHB) and/or Advanced System Bus (ASB) may be used, as may the Handspring Interconnect specified by Handspring, Inc. (Mountain View, CA). Still further, code translator 22 may be connected to memory 16 using a Unified Memory Architecture connection (e.g. see Fig. 1A). The bus used for the Unified Memory Architecture connection may be any of the above buses. In other alternatives, code translator 22 may be directly connected to CPU 12 or memory 16, or may be integrated into CPU 12, memory controller 14, or PCI bridge 18.

As used herein, a "terminating condition" is any event which causes code translator 22 to terminate translation of a code sequence. The events which cause termination may be varied from embodiment to embodiment, according to design choice.

As used herein, the term "translation" refers to generating one or more instructions in a second instruction

set which provide the same result, when executed, as a first instruction in a first instruction set. For example, the one or more instructions may perform the same operation or operations on the operands of the first instruction to generate the same result the first instruction would have generated. Additionally, the one or more instructions may have the same effect on other architect state as the first instruction would have had.

As used herein, the term "code sequence" refers to an ordered set of one or more instructions. A "source code sequence" is a code sequence using instructions in a first instruction set which may be translated by code translator 22, and a "translated code sequence" is a code sequence using instructions in a second instruction set and which is the translation of at least a portion of the source code sequence. For example, in system 10, the source code sequence may be coded using non-native instructions and the translated code sequence may be coded using native instructions.

As used herein, a "branch target address" is an address generated during execution of a branch instruction, which identifies the first instruction of the target path of the branch instruction. The branch target address may be generated in a variety of ways. For example, the branch target address may be directly specified in the branch instruction. Alternatively, the branch target address may be generated by adding a displacement and/or other operands of the branch instruction to the address of the branch instruction (or the address of the next sequential instruction to the branch instruction).

As used herein, the "sequential path" of a conditional branch instruction comprises the instruction sequence executed if the branch instruction is not taken (i.e. the condition is not met). The sequential path generally includes instructions which are at addresses following the conditional branch instruction in order. The "target path" of a branch instruction is the instruction sequence located at the branch target address.

Turning next to Fig. 2, a block diagram of one embodiment of code translator 22 is shown. Other embodiments are possible and contemplated. In the illustrated embodiment, code translator 22 may include a fetch circuit 30, a code generator circuit 32, a branch resolution buffer (BRB) 34, a BRB control circuit 36, a BRB monitor circuit 38, and a write buffer 40. Fetch circuit 30 is coupled to fetch source instructions from memory 16, and is coupled to code generator circuit 32 and BRB monitor circuit 38. Code generator circuit 32 is further coupled to write buffer 40, BRB control circuit 36, branch resolution buffer 34, and BRB monitor circuit 38. BRB monitor circuit 38 is further coupled to branch resolution buffer 34 and BRB control circuit 36, which is still further coupled to branch resolution buffer 34. Branch resolution buffer 34 is still further coupled to write buffer 40, which is coupled to provide the translated code sequence to memory 16.

Generally, fetch circuit 30 may fetch the source code sequence beginning at the source address provided by CPU 12. Fetch circuit 30 may include a buffer for storing source instructions until they can be translated by code generator circuit 32. Fetch circuit 30 may proceed to fetch sequentially through the source code sequence, unless redirected by fetch control information from code generator circuit 32. Fetch circuit 30 provides source code and addresses to code generator circuit 32, which generates corresponding translated code and addresses and stores the translated code and addresses in write buffer 40. The translated code sequence may be written to memory 16 in any suitable manner. Code generator circuit 32 may be configured to concurrently translate two or more source instructions, if desired. In one embodiment, code generator circuit 32 may concurrently translate up to 4 source instructions.

The circuitry illustrated in Fig. 2 also handles conditional branch instructions. First discussed will be an

embodiment which handles forward conditional branch instructions. Backward conditional branch instructions may be handled by inserting a conditional exit instruction for the taken case, where the condition for the conditional exit instruction may be the same as the backward conditional branch instruction's condition, and translation may continue sequentially. Alternatively, the backward conditional branch instruction may be predicted and the predicted path translated. The conditional exit instruction may test for the condition, or the logical opposite of the condition, corresponding the backward conditional branch instruction depending on the predicted direction. In one particular implementation, if the backward conditional branch instruction targets the first instruction in the source code sequence, a translated conditional branch instruction to the first instruction in the translated code sequence may be generated instead of a conditional exit instruction. Since backward conditional branch instructions are often associated with loops, it may be desirable to have a separate translated code sequence for the loop. Alternative embodiments are discussed further below.

If code generator circuit 32 detects a forward conditional branch instruction, code generator circuit 32 signals BRB control circuit 36 and provides information corresponding to the conditional branch instruction to branch resolution buffer 34. Additionally, code generator circuit 32 generates a translated conditional branch instruction for the translated code sequence and provides it to write buffer 40. BRB control circuit 36 allocates an entry in branch resolution buffer 34 to store the information, and causes branch resolution buffer 34 to update the allocated entry with the information provided by code generator circuit 32. More particularly, code generator circuit 32 may provide at least the address within the translated code sequence of the translated conditional branch instruction (the "branch address") and the branch target address of the source conditional branch instruction within the source code sequence. Code generator circuit 32 may continue translating down the sequential path, and thus the branch target address is the address of the path not currently being translated.

Since the conditional branch instruction is forward, the sequential path and the target path may merge at some point in the source code sequence. Accordingly, the branch target address stored in branch resolution buffer 34 may match a subsequent address to be translated due to sequential fetching by fetch circuit 30. Furthermore, in one embodiment, code translator 32 may detect other terminating conditions in the sequential path and may redirect fetch circuit 30 to fetch the branch target address of the forward conditional branch instruction. In either of these cases, one or more source instructions at the branch target address of the forward conditional branch instruction may be translated. BRB monitor circuit 38 monitors the branch target addresses stored in each allocated entry of branch resolution buffer 34 and the source addresses of source instructions being provided by fetch circuit 30 to code generation circuit 32 for translation. If BRB monitor circuit 38 detects a match between a branch target address and one of the source addresses, then the target path of the corresponding conditional branch is being translated. BRB monitor 38 signals code generator 32 and BRB control circuit 36 of the match (illustrated as BRB hit in Fig. 2). If branch resolution buffer 34 includes multiple entries, BRB monitor circuit 38 also identifies the entry for which the match is detected.

In response to the BRB hit, BRB control circuit 36 causes branch resolution buffer 34 to output the branch address from the hitting entry to write buffer 40. Additionally, code generator circuit 32 may generate an indication of the branch target address to be coded into the translated conditional branch instruction in response to the signals from BRB monitor circuit 38. In this manner, the translated conditional branch instruction is coded to generate a branch target address, during execution, which identifies the first translated instruction in the target path.

For example, the indication may be a displacement from the branch address to the branch target address. The displacement may be added to the branch address during execution to generate the branch target address. Alternatively, the indication may be the branch target address to be encoded as an absolute conditional branch instruction. Any method for generating the branch target address (and thus specifying the indication to be generated by code generator circuit 32) may be used. Write buffer 40 looks up the branch address to identify the translated conditional branch instruction for which the target path is being translated, and updates the translated conditional branch with the indication provided by code generator circuit 32. Additionally, since the target path of the translated conditional branch instruction has been located in the translated code sequence and the translated conditional branch instruction has been coded to generate the branch target address of that target path, the entry in branch resolution buffer 34 may be invalidated.

In one embodiment, when code generator circuit 32 initially translates a conditional branch instruction in the source code sequence, code generator circuit 32 generates a conditional exit instruction as the translated conditional branch instruction. In this manner, if the target path is not translated prior to terminating translation, the translated code sequence operates properly because the conditional exit instruction exits the translated code sequence if the branch instruction is taken. Additionally, if branch resolution buffer 34 overflows and an entry allocated to a particular conditional branch is to be overwritten, the entry may be overwritten without additional action to the translated code sequence. Alternatively, the translated conditional branch instruction may not be coded to generate a specific target address during the initial translation. In such an embodiment, coding the conditional exit instruction may be deferred until the translation is terminated or the entry corresponding to the translated conditional branch is to be overwritten.

Generally, branch resolution buffer 34 may include any number of entries. In one specific embodiment, branch resolution buffer 34 may have a single entry which may be overwritten each time a conditional branch instruction is encountered and recorded in branch resolution buffer 34. For such an embodiment, BRB control circuit 36 may set the valid indication in the entry to a valid state in response to the conditional branch signal from code generation circuit 32 and set the valid indication in the entry to an invalid state in response to a BRB hit signalled by BRB monitor circuit 38. The branch address may be output from the single entry. Other embodiments may include multiple entries, and BRB control circuit 36 may employ an allocation scheme among the entries for various conditional branch instructions.

While the above description of Fig. 2 has described an embodiment which handles forward conditional branch instructions and continues translation first with the sequential path and subsequently with the target path, other embodiments may also handle backward conditional branch instructions in a similar manner. In the case of a backward conditional branch instruction, following the sequential path first may not generally lead to a merge with the branch target address. However, if another terminating condition is reached on the sequential path, code generator circuit 32 may translate instructions down the target path of the backward conditional branch instruction. This may have the effect of partially unrolling a loop, if the backward conditional branch instruction is a loop terminator. Furthermore, other embodiments may continue translation with the target path first, and then return to the sequential path, for either forward or backward conditional branch instructions. For such an embodiment, the sequential address in the source code sequence would be stored in branch resolution buffer 34 instead of the branch target address. Additionally, the translated conditional branch instruction would test for the opposite logical

condition of the source conditional branch instruction, since the target path instructions in the translated code sequence would follow the translated conditional branch instruction and the sequential instructions would be placed at the target of the translated conditional branch instruction.

It is noted that, in some embodiments, code generator circuit 32 may generate additional instructions prior to the translated conditional branch instruction ("restore code") and/or prior to the first instruction in the target path within the translated code sequence ("merge code"). More particularly, if code translator 22 employs various translation optimizations, restore code and/or merge code may be used to ensure that the optimizations remain transparent. For example, code translator 22 may be responsible for allocating registers to variables in the translated code sequence. If code translator 22 reallocates a register which was allocated to a first variable prior to a conditional branch instruction to a second variable in the sequential path, that register may need to be returned to allocation to the first variable in the target path. If the second variable is not merely a temporary variable, it may need to be committed to a memory location as well. Merge code may be used to perform these operations. As another example, in an embodiment used to translate Java instructions to native instructions, updates to the stack pointer and the program counter may be deferred. Since, in one implementation, the conditional branch instruction is initially translated as an exit instruction, restore code may be used to update the stack pointer and program counter with the deferred updates. Similarly, merge code may be used to commit the updates to the stack pointer and the program counter which occurred within the sequential path before merging the sequential path with the target path. In this manner, no deferred updates are outstanding at the first instruction in the target path, regardless of whether the target path or the sequential path is actually taken during execution. Generally speaking, restore code and/or merge code may be used to synchronize the optimization state of code translator 22 at a merge point of two paths (as may be the case for a forward branch) or to synchronize the optimization state prior to the branch to the optimization state prior to translating down the other path (as may be the case for a backward branch).

Generally, code generator circuit 32 may be configured to translate each source instruction to one or more translated instructions, or to generate an exit instruction if the source instruction is not translated by code generator circuit 32. Some source instructions may be too complex and/or too infrequent to invest the hardware for translating them, and these source instructions may be handled via emulation in the control program. More particularly, code generator circuit 32 may comprise a programmable logic array (PLA) structure, combinatorial logic, or a lookup table (either a read-only memory (ROM) lookup table, or a random access memory (RAM) lookup table). In lookup table form, each source instruction could be assigned an entry in the table, with the corresponding set of translated instructions stored in the entry.

It is noted that, while the above description has referred to capturing information for each conditional branch instruction in branch resolution buffer 34, other embodiments may filter certain conditional branch instructions out. For example, a forward conditional branch instruction having a branch target address which is not near the branch address of that conditional branch instruction may be less likely to experience a merge of the sequential and target paths than is another forward conditional branch instruction having a branch target address which is near the branch address. Thus, a maximum displacement of the conditional branch instruction may be used to filter out conditional branch instructions with a large displacement. The filtered conditional branch instructions may be translated as conditional exit instructions.

It is further noted that, while BRB monitor circuit 38 is used in the present embodiment to detect

translation of the target path (or the sequential path, if the target path is translated first), other embodiments may detect translation of the target path in other ways. For example, code generator 32 may begin translation of the target path when it detects various termination conditions and may update the translated conditional branch instruction to generate the branch target address in response to detecting the termination condition (and may fetch the target path).

It is still further noted that, while write buffer 40 is used in the present embodiment, other embodiments may write translated code sequences to memory as they are generated. Additionally, updating the translated conditional branch instructions as described above may also be performed via a write to the corresponding memory location. It is also noted that, while the above illustration shows BRB monitor circuit 38 receiving source addresses to monitor from fetch circuit 30, BRB monitor circuit 38 may receive the source addresses from any source (including, e.g. code generator circuit 32).

It is still further noted that the above description refers to selecting one of the target path or the sequential path and translating the selected path to a terminating condition, followed by translation of one or more instructions from the other path and coding the translated conditional branch instruction to generate a target address identifying the other path in the translated code sequence. However, other embodiments may employ other methods. For example, an embodiment is contemplated in which a target address within the translated address space is selected for the translated conditional branch instruction during translation of the conditional branch instruction. The translated conditional branch instruction may be coded to generate the selected target address. The selected target address may allow, for example, a predetermined number of sequential path instructions to be inserted between the branch instruction and the selected target address. In such an embodiment, the predetermined number of sequential path instructions may be translated, followed by a return instruction, followed by target path instructions. Still further, target path and sequential path instructions may be translated simultaneously in such a case. The branch resolution buffer may be eliminated from embodiments which code the selected target address during the translation of the conditional branch instruction.

As used herein, the term "forward branch instruction" refers to a branch instruction (conditional or unconditional) for which the branch target address is numerically larger than the branch address of the branch instruction. The term "backward branch instruction" refers to a branch instruction (conditional or unconditional) for which the branch target address is numerically smaller than the branch address of the branch instruction.

Turning next to Fig. 3, a block diagram of one embodiment of a branch resolution buffer entry 50 is shown. Other embodiments are possible and contemplated. In the illustrated embodiment, buffer entry 50 includes a valid indication 52, a branch address field 54, a branch target address field 56, and an optional state field 58.

Valid indication 52 indicates whether or not information in the buffer entry is valid (e.g. corresponds to a conditional branch instruction). The valid indication may, for example, be a bit indicative, when set, that the information is valid and indicative, when clear, that the information is not valid. Alternatively, the bit being clear may indicate validity and the bit being set may indicate invalidity. Other encodings involving two or more bits may be used, if desired.

Branch address field 54 may store the branch address of the corresponding translated conditional branch instruction within the translated code sequence. The branch address may be used to locate the translated conditional branch instruction in the translated code sequence to code the branch target address if the target path is

translated.

Branch target address field 56 may store the branch target address of the corresponding conditional branch instruction within the source code sequence. Branch target address field 56 may be monitored by BRB monitor circuit 38 to detect when the target path is being translated. Alternatively, the sequential address may be stored in embodiments which translate the target path first.

Finally, state information field 58 may be used to store miscellaneous state information that may be useful in various embodiments. For example, information about the type of branch instruction may be stored. Additionally, information regarding the optimization state of code translator 22 at the time of translation of the corresponding conditional branch instruction may be stored, to aid in merge code generation, if applicable.

Turning next to Fig. 4, a flowchart is shown illustrating operation of one embodiment of code translator 22 in translating an instruction. Other embodiments are possible and contemplated. While the steps shown are illustrated in a particular order for ease of understanding, any suitable order may be used. Furthermore, the steps may be performed in parallel by combinatorial logic circuitry within code translator 22. Still further, various steps may be performed in different clock cycles, as desired.

Code translator 22 may determine if the instruction is a conditional branch (decision block 60). Depending on the embodiment, the determination of decision block 60 may be any conditional branch instruction, only forward conditional branch instructions, only backward conditional branch instructions, only conditional branch instructions have a branch target address within a predefined distance of the conditional branch instruction, etc. If the instruction is not a conditional branch instruction, code translator 22 may translate the instruction to one or more instructions in the translated code sequence (block 62).

On the other hand, if the instruction is a conditional branch instruction, code translator 22 may optionally generate restore code (block 64) if code translator 22 employs optimizations which require such code, as described above. Additionally, code translator 22 may generate a conditional exit instruction corresponding to the conditional branch instruction for the translated code sequence (block 66). As mentioned above, alternatively a conditional branch instruction may be generated and subsequently updated with a branch target address (if the target path is translated) or recoded as a conditional exit instruction (if the target path is not translated). Finally, code translator 22 may allocate a branch resolution buffer entry for the conditional branch instruction (block 68). As mentioned above, in one embodiment the branch resolution buffer may be a single entry, and allocating an entry may comprise overwriting the single entry with information for the conditional branch instruction. For multiple entry branch resolution buffers, any allocation scheme may be used. An exemplary scheme is illustrated in Fig. 8 below. Additionally, other schemes may be used. For example, the entries may be treated as a first-in, first-out (FIFO) buffer or a stack. According to one embodiment of code translator 22 illustrated in Fig. 2, blocks 60, 62, 64, and 66 may be performed by code generation circuit 32 and block 68 may be performed by BRB control circuit 36 (or may be implied if branch resolution buffer 34 has one branch resolution buffer entry).

Turning next to Fig. 5, a flowchart is shown illustrating operation of one embodiment of code translator 22 for detecting that translation of the target path of a conditional branch instruction is occurring. Other embodiments are possible and contemplated. While the steps shown are illustrated in a particular order for ease of understanding, any suitable order may be used. Furthermore, the steps may be performed in parallel by combinatorial logic circuitry within code translator 22. Still further, various steps may be performed in different clock cycles, as

desired.

Code translator 22 determines if a terminating condition is detected for a source instruction (other than the merging of a target path and a sequential path -- decision block 70). If there is a terminating condition, code translator 22 generates an exit instruction in the translated code sequence (block 72). Additionally, code translator 22 determines if there is a valid branch resolution buffer entry (decision block 74). If there is a valid branch resolution buffer entry, code translator 22 may fetch the branch target address (from that branch resolution buffer entry) in the source instruction sequence and translate instructions from the target path indicated by that branch target address (block 76). It is noted that, if more than one branch resolution buffer entry is valid, any entry may be selected. For example, the entry corresponding to the most recently encountered branch instruction may be selected. Alternatively, the entry corresponding to the oldest branch instruction may be selected (i.e. the least recently encountered branch instruction). In yet another alternative, a random entry may be selected. Code translator 22 changes the conditional exit instruction stored at the branch address in the selected entry to a local branch instruction (i.e. a branch instruction having a branch target address within the translated code sequence). More specifically, the branch target address of the local branch instruction is the address within the translated code sequence at which the first instruction of the translated target path is to be stored (block 78). The first instruction of the target path may be stored sequential to the exit instruction generated at block 72, for this case. Additionally, since the conditional branch instruction corresponding to the selected branch resolution buffer entry is resolved, the selected branch resolution buffer entry may be invalidated (block 80).

Code translator 22 may also determine if the branch target address in any valid branch resolution buffer entry matches the current source address of an instruction to be translated (decision block 82). It is noted that decision blocks 70 and 84 are independent and may be performed in parallel. If the branch target address of a branch resolution buffer entry does match the current source address, merge code may optionally be generated (as discussed above -- block 84). Additionally, the conditional exit instruction stored at the branch address of the selected branch resolution buffer entry is changed to a local branch instruction (block 78) with the branch target address coded to the address within the translated code sequence of the first instruction from the translated target path. More particularly, the first instruction from the target path may be stored sequential to the last translated instruction before the current source address (and any merge code, if applicable) in this case. Additionally, the selected branch resolution buffer entry is invalidated (block 80). According to one embodiment of code translator 22 illustrated in Fig. 2, blocks 70, 72, 74, 76, and 84 may be performed by code generation circuit 32, block 78 may be performed by a combination of code generation circuit 32, BRB control circuit 36, and branch resolution buffer 34, and block 82 may be performed by BRB monitor circuit 38.

Turning now to Fig. 6, an example source code sequence and corresponding translated code sequence is shown according to one embodiment of code translator 22. The example source code sequence is shown at reference numeral 90, an intermediate translated code sequence during the translation is shown at reference numeral 92, and a final translated code sequence is illustrated at reference numeral 94. Source code sequence 90 includes three blocks of instructions (S_B1, S_B2, and S_B3). Each instruction in source code sequence 90 is labeled with a capital S (for source), followed by an underscore, followed by an identifier for that instruction. Additionally, each instruction is assigned a representative address (SA for S_inst0, SA+1 for S_inst1, etc.). The address labels are merely to indicate order between the instructions and not necessarily the exact number of bytes between addresses.

Block S_B1 includes the first few source instructions including a compare instruction (S_cmp) to generate the condition for a conditional branch instruction (S_bc) which terminates block S_B1. Block S_B2 comprises the sequential path of the conditional branch instruction and Block S_B3 comprises the target path (into which the sequential path merges). The branch target address of the conditional branch instruction (S_bc) is SA+7, the first instruction in block S_B3. Similarly, translated instructions are each labeled with a capital T (for translated), followed by an underscore, followed by an identifier indicating to which source instruction that translated instruction corresponds, followed by another underscore, followed by a sequence number (since a given source instruction may be translated into more than one translated instruction). For example, translated instructions T_inst0_0 and T_inst0_1 may be two instructions generated in response to source instruction S_inst0. Also each translated instruction is assigned a representative address (TA for T_inst0_0, TA+1 for T_inst0_1, etc.).

Intermediate translated code sequence 92 illustrates a point in the translation after the conditional branch (S_bc) which terminates block S_B1 is translated to a translated conditional branch (T_bc) which terminates block T_B1 but before the merge point in the sequential and target paths of the conditional branch (S_bc) has been reached. The merge point is address SA+7, instruction S_inst5. Accordingly, the translated conditional branch (T_bc) in intermediate code sequence 92 is coded with the predefined target address which exits to the control program. In other words, the translated conditional branch is coded as a conditional exit instruction. The exiting of the sequence is illustrated by arrow 96. Intermediate translated code sequence 92 also includes two instructions in block T_B2, corresponding to instruction S_inst2 in block S_B2.

Additionally, the contents of the corresponding branch resolution buffer entry are illustrated below intermediate translated code sequence 92. The branch resolution buffer entry is valid (since the translated conditional branch T_bc remains unresolved at this point). The branch address field indicates the branch address of the translated conditional branch (TA+4). Additionally, the target address field indicates the branch target address of the original conditional branch in the source sequence (SA+7).

Final translated code sequence 94 is the translated code sequence on completion of the translation of source code sequence 90. Accordingly, final translated code sequence 94 includes three instruction blocks T_B1, T_B2, and T_B3. Block T_B1 is terminated by the translated conditional branch instruction (T_bc). Additionally, since the branch target address SA+7 was reached in the translation, the translated conditional branch instruction (T_bc) is coded with to generate the branch target address indicating the first instruction of the target path (TA+9, instruction T_inst5_0). The resolving of the branch target address of the translated conditional branch is further illustrated by arrow 98. Block T_B2 includes translated instructions corresponding to the source instructions in block S_B2, and block T_B3 includes translated instructions corresponding to source instructions in block S_B3. Additionally, since the branch target address of the conditional branch instruction (S_bc) was reached during the translation, the branch resolution buffer entry is invalid.

As final translated code sequence 94 illustrates, both the sequential and target paths of the conditional branch S_bc have been translated. Accordingly, final translated code sequence 94 may be executed in both the taken and not taken cases of conditional branch instruction S_bc. Conditional branch instruction S_bc may therefore be efficiently handled by code translator 22.

While the example of Fig. 6 illustrates a single conditional branch, sequences including multiple conditional branches may be handled as well. For example, if block S_B3 were terminated by a conditional branch,

even a branch resolution buffer having a single entry could handle both branches. Furthermore, a branch resolution buffer having multiple entries may handle multiple nested conditional branches, even if the nested branches have the same branch target address (i.e. have the same merge point).

The example of Fig. 6 does not show any merge code. However, various embodiments may include merge code within block T_B2, if needed.

Turning next to Fig. 7, a second example source code sequence and corresponding translated code sequence is shown. The example of Fig. 7 uses the same labeling conventions for source instructions, source addresses, translated instructions, and destination addresses as the example of Fig. 6 uses. More particularly, the example of Fig. 7 includes a source code sequence 100, an intermediate translated code sequence 102, and a final translated code sequence 104. Source code sequence 100 is similar to source code sequence 90, except that S_inst3 in block S_B2 is not translatable by code translator 22 (indicated by the parenthetical NT in Fig. 7).

Intermediate translated code sequence 102 illustrates a point in the translation after the conditional branch (S_bc) which terminates block S_B1 is translated to a translated conditional branch (T_bc) which terminates block T_B1 and after the untranslatable instruction S_inst3 has been attempted. Accordingly, since the target address of the conditional branch instruction (S_bc) has not been reached, the translated conditional branch (T_bc) in intermediate code sequence 102 is coded with the predefined target address which exits to the control program. In other words, the translated conditional branch is coded as a conditional exit instruction. The exiting of the sequence is illustrated by arrow 106. Intermediate translated code sequence 102 also includes two instructions in block T_B2, corresponding to instruction S_inst2 in block S_B2. Furthermore, intermediate translated code sequence 102 includes an exit instruction (unconditional branch T_B) coded with the predefined exit address. The exit instruction is inserted in response to the attempt to translate source instruction S_inst3. Again, the exiting of intermediate code sequence 102 by the exit instruction is illustrated by arrow 108.

Additionally, the contents of the corresponding branch resolution buffer entry are illustrated below intermediate translated code sequence 102. The branch resolution buffer entry is valid (since the translated conditional branch T_bc remains unresolved at this point). The branch address field indicates the branch address of the translated conditional branch (TA+4). Additionally, the target address field indicates the branch target address of the original conditional branch in the source sequence (SA+7).

Since the sequential path (block T_B2) has been terminated by an untranslatable instruction, code translator 22 may return to the target path and translate instructions from the target path and resolve the translated conditional branch instruction (T_bc). Final translated code sequence 104 illustrates the resulting completed sequence.

Since instructions from the target path have been translated, the translated conditional branch instruction (T_bc) is coded to generate the target address of the first instruction in the target path (TA+8, instruction T_inst5_0). The resolution of the conditional branch instruction (T_bc) is further illustrated by arrow 110. Thus, even though there is no merge between the sequential and target paths in this case, instructions from each path are still included in the final translated code sequence 104. Again, the conditional branch instruction (S_bc) may be efficiently handled by final translated code sequence 104.

Turning next to Fig. 8, a flowchart illustrating an exemplary allocation scheme which may be used by BRB control circuit 36 to allocate a branch resolution buffer entry for a conditional branch instruction when branch

resolution buffer 34 includes two entries is shown. Other embodiments are possible and contemplated. While the steps shown are illustrated in a particular order for ease of understanding, any suitable order may be used. Furthermore, the steps may be performed in parallel by combinatorial logic circuitry within BRB control circuit 36. Still further, various steps may be performed in different clock cycles, as desired.

The flowchart of Fig. 8 illustrates using one of the two entries for the oldest conditional branch instruction (the least recently encountered) in the current source code sequence and using the other entry for the most recently encountered conditional branch instruction. The most recently encountered conditional branch instruction may be the one which is most likely, in many code sequences, to experience a merging of the target path and the sequential path. The oldest conditional branch instruction may, in many code sequences, be the most likely to be taken (and thus the most useful to have instructions from the target path translated).

If the oldest entry is not occupied (decision block 130), then the conditional branch being allocated a branch resolution buffer entry is the first encountered in the current source code sequence or is the first encountered after the target path of the oldest conditional branch instruction has been translated. Accordingly, the oldest entry may be allocated to the branch (block 132). Alternatively, if the target path of the oldest conditional branch instruction has been translated, the contents of the other entry may be copied to the oldest entry (or the other entry may be indicated as the oldest entry).

If the oldest entry is occupied, the other entry (for the most recently encountered conditional branch instruction) may be allocated (block 134). If the most recent entry is occupied, the contents are overwritten by the information for the newly encountered conditional branch instruction.

While the flowchart of Fig. 8 illustrates two entries, with one being used for the oldest conditional branch instruction and the other for the most recently encountered branch instruction, the allocation scheme may be extended to more than two entries. For example, one entry could be used for the oldest conditional branch instruction and the remaining entries could be used for the N most recently encountered conditional branch instructions. The remaining entries could be allocated in a FIFO fashion, for example. In another alternative, M of the entries could be used for the oldest conditional branch instructions and N of the entries could be used for the most recently encountered conditional branch instructions. Other allocation schemes (such as a FIFO or stack) may be used as well.

Turning next to Fig. 9, a block diagram of a carrier medium 120 including a database representative of code translator 22 is shown. Generally speaking, a carrier medium may include storage media such as magnetic or optical media, e.g., disk or CD-ROM, volatile or non-volatile memory media such as RAM (e.g. SDRAM, RDRAM, SRAM, etc.), ROM, etc., as well as transmission media or signals such as electrical, electromagnetic, or digital signals, conveyed via a communication medium such as a network and/or a wireless link.

Generally, the database of code translator 22 carried on carrier medium 120 may be a database which can be read by a program and used, directly or indirectly, to fabricate the hardware comprising code translator 22. For example, the database may be a behavioral-level description or register-transfer level (RTL) description of the hardware functionality in a high level design language (HDL) such as Verilog or VHDL. The description may be read by a synthesis tool which may synthesize the description to produce a netlist comprising a list of gates in a synthesis library. The netlist comprises a set of gates which also represent the functionality of the hardware comprising system 10. The netlist may then be placed and routed to produce a data set describing geometric shapes

to be applied to masks. The data set, for example, may be a GDSII (General Design System, second revision) data set. The masks may then be used in various semiconductor fabrication steps to produce a semiconductor circuit or circuits corresponding to system 10. Alternatively, the database on carrier medium 120 may be the netlist (with or without the synthesis library) or the data set, as desired.

While carrier medium 120 carries a representation of code translator 22, other embodiments may carry a representation of any portion of code translator 22, as desired, including any code generator circuits, monitor circuits, control circuits, buffers), etc. Furthermore, carrier medium 120 may carry a representation of system 10 or any portion thereof.

Numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.

WHAT IS CLAIMED IS:

1. An apparatus comprising a first circuit configured to translate a first conditional branch instruction in a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions in a second instruction set, wherein said first circuit is configured to translate one or more instructions from a first path of a target path or a sequential path of said first conditional branch instruction, and wherein said first circuit is configured to translate at least a first instruction within a second path of said target path or said sequential path to a second instruction in said translated code sequence, and wherein said first circuit is configured to code said second conditional branch instruction to generate a branch target address, during execution, which identifies said second instruction in said translated code sequence.

2. The apparatus as recited in claim 1 wherein said first circuit is configured to output a first address of said second conditional branch instruction within said translated code sequence and a second address of said first instruction within said second path, and wherein the apparatus further comprises a buffer including at least a first entry, said first entry configured to store said first address and said second address.

3. The apparatus as recited in claim 2 wherein said buffer comprises a plurality of entries including said first entry, wherein each of said plurality of entries is configured to store addresses corresponding to different branches.

4. The apparatus as recited in claim 2 wherein said buffer comprises a second entry, and wherein said first entry stores addresses corresponding to a most recently encountered conditional branch instruction, and wherein said second entry stores addresses corresponding to an oldest conditional branch instruction.

5. The apparatus as recited in claim 1 wherein said first circuit is configured to translate at least said first instruction located by said second address within said second path to a second instruction in said translated code sequence responsive to a terminating condition in said first path.

6. The apparatus as recited in claim 5 further comprising a second circuit coupled to said buffer and configured to monitor addresses of instructions from said source code sequence which are to be translated, wherein said second circuit is configured to detect if one of said addresses matches said second address, and wherein said second circuit detecting a match comprises said terminating condition.

7. The apparatus as recited in claim 1 wherein said first circuit is configured, during translating said first conditional branch instruction to said second conditional branch instruction, to code said second conditional branch instruction to generate a first predefined branch target address during execution.

8. The apparatus as recited in claim 7 wherein said first predefined branch target address is a return to a control program which initiates translation of said source code sequence to said translated code sequence.

9. The apparatus as recited in claim 5 wherein said first circuit is configured to generate a third branch instruction responsive to detecting said terminating condition, said third branch instruction coded to generate a first predefined branch target address during execution.

10. The apparatus as recited in claim 9 wherein said first predefined branch target address is a return to a control program which initiates translation of said source code sequence to said translated code sequence.

11. The apparatus as recited in claim 1 wherein said first conditional branch instruction is a forward branch instruction.

12. The apparatus as recited in claim 1 wherein said first circuit is configured to generate one or more instructions prior to said second instruction in said translated code sequence, wherein said one or more instructions synchronize said target path and said sequential path in said translated code sequence.

13. A method comprising:

translating a first conditional branch instruction from a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions defined in a second instruction set;

translating down a first path of a target path or a sequential path of said first conditional branch instruction;

translating at least a first instruction of a second path of said target path or said sequential path of said first conditional branch instruction to a second instruction in said translated code sequence; and

coding said second conditional branch instruction to generate a branch target address, during execution, which identifies said second instruction.

14. The method as recited in claim 13 further comprising:

recording a first address, said first address locating said second conditional branch instruction in said translated code sequence;

recording a second address, said second address locating the first instruction of the second path;

15. The method as recited in claim 13 further comprising:

detecting a termination condition in said first path, said translating at least said first instruction being responsive to said detecting.

16. The method as recited in claim 15 wherein said detecting said terminating condition comprises monitoring addresses of instructions to be translated to detect a match between one of said addresses and said second address.

17. The method as recited in claim 15 further comprising generating a third branch instruction responsive to said detecting said terminating condition, said third branch instruction coded to generate a first predefined branch target

address during execution.

18. The method as recited in claim 17 wherein said first predefined branch target address is a return to a control program which initiates translation of said source code sequence to said translated code sequence.

19. The method as recited in claim 13 wherein said translating said first conditional branch instruction to said second conditional branch instruction comprises coding said second conditional branch instruction to generate a first predefined branch target address during execution.

20. The method as recited in claim 13 further comprising generating one or more instructions prior to said second instruction in said translated code sequence, wherein said one or more instructions synchronize said target path and said sequential path in said translated code sequence.

21. A carrier medium configured to hold a database representing : a first circuit configured to translate a first conditional branch instruction in a source code sequence coded using instructions defined in a first instruction set to a second conditional branch instruction in a translated code sequence coded using instructions in a second instruction set, wherein said first circuit is configured to translate one or more instructions from a first path of a target path or a sequential path of said first conditional branch instruction, and wherein said first circuit is configured to translate at least a first instruction within a second path of said target path or said sequential path to a second instruction in said translated code sequence, and wherein said first circuit is configured to code said second conditional branch instruction to generate a branch target address, during execution, which identifies said second instruction in said translated code sequence.

22. The carrier medium as recited in claim 21 wherein said first circuit is configured to output a first address of said second conditional branch instruction within said translated code sequence and a second address of said first instruction within said second path, and wherein the database further represents a buffer including at least a first entry, said first entry configured to store said first address and said second address.

23. The carrier medium as recited in claim 22 wherein said buffer comprises a plurality of entries including said first entry, wherein each of said plurality of entries is configured to store addresses corresponding to different branches.

24. The carrier medium as recited in claim 22 wherein said buffer comprises a second entry, and wherein said first entry stores addresses corresponding to a most recently encountered conditional branch instruction, and wherein said second entry stores addresses corresponding to an oldest conditional branch instruction.

25. The carrier medium as recited in claim 21 wherein said first circuit is configured to translate at least said first instruction located by said second address within said second path to a second instruction in said translated code sequence responsive to a terminating condition in said first path.

26. The carrier medium as recited in claim 25 wherein the database further represents a second circuit coupled to said buffer and configured to monitor addresses of instructions from said source code sequence which are to be translated, wherein said second circuit is configured to detect if one of said addresses matches said second address, and wherein said second circuit detecting a match comprises said terminating condition.

27. The carrier medium as recited in claim 21 wherein said first circuit is configured, during translating said first conditional branch instruction to said second conditional branch instruction, to code said second conditional branch instruction to generate a first predefined branch target address during execution.

28. The carrier medium as recited in claim 27 wherein said first predefined branch target address is a return to a control program which initiates translation of said source code sequence to said translated code sequence.

29. The carrier medium as recited in claim 25 wherein said first circuit is configured to generate a third branch instruction responsive to detecting said terminating condition, said third branch instruction coded to generate a first predefined branch target address during execution.

30. The carrier medium as recited in claim 29 wherein said first predefined branch target address is a return to a control program which initiates translation of said source code sequence to said translated code sequence.

31. The carrier medium as recited in claim 21 wherein said first conditional branch instruction is a forward branch instruction.

32. The carrier medium as recited in claim 21 wherein said first circuit is configured to generate one or more instructions prior to said second instruction in said translated code sequence, wherein said one or more instructions synchronize said target path and said sequential path in said translated code sequence.

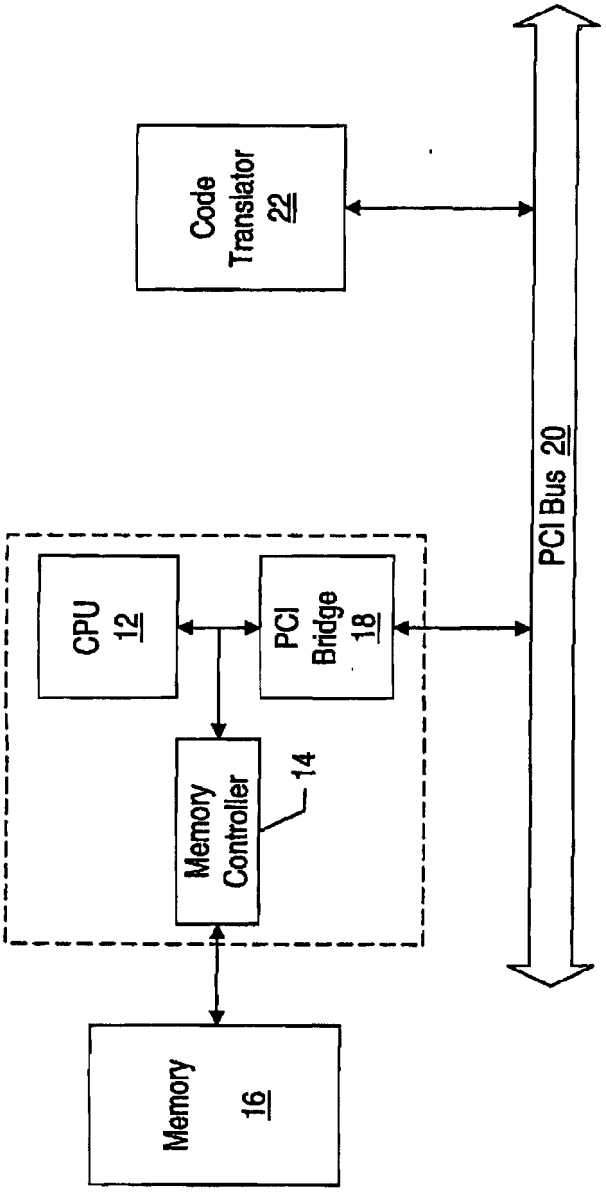

Code Translator 22
PCI Bus 20
CPU 12
PCI Bridge 18
Memory Controller
14
Memory 16
10

Fig. 1

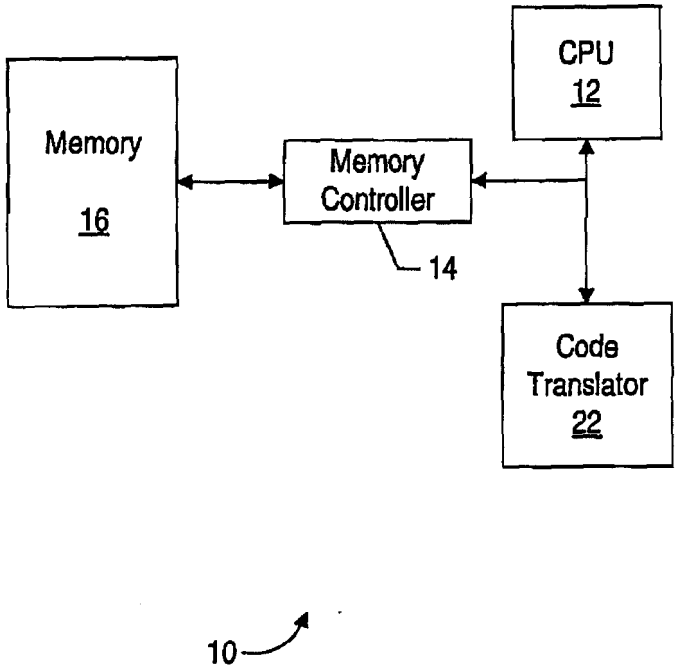


Fig. 1A

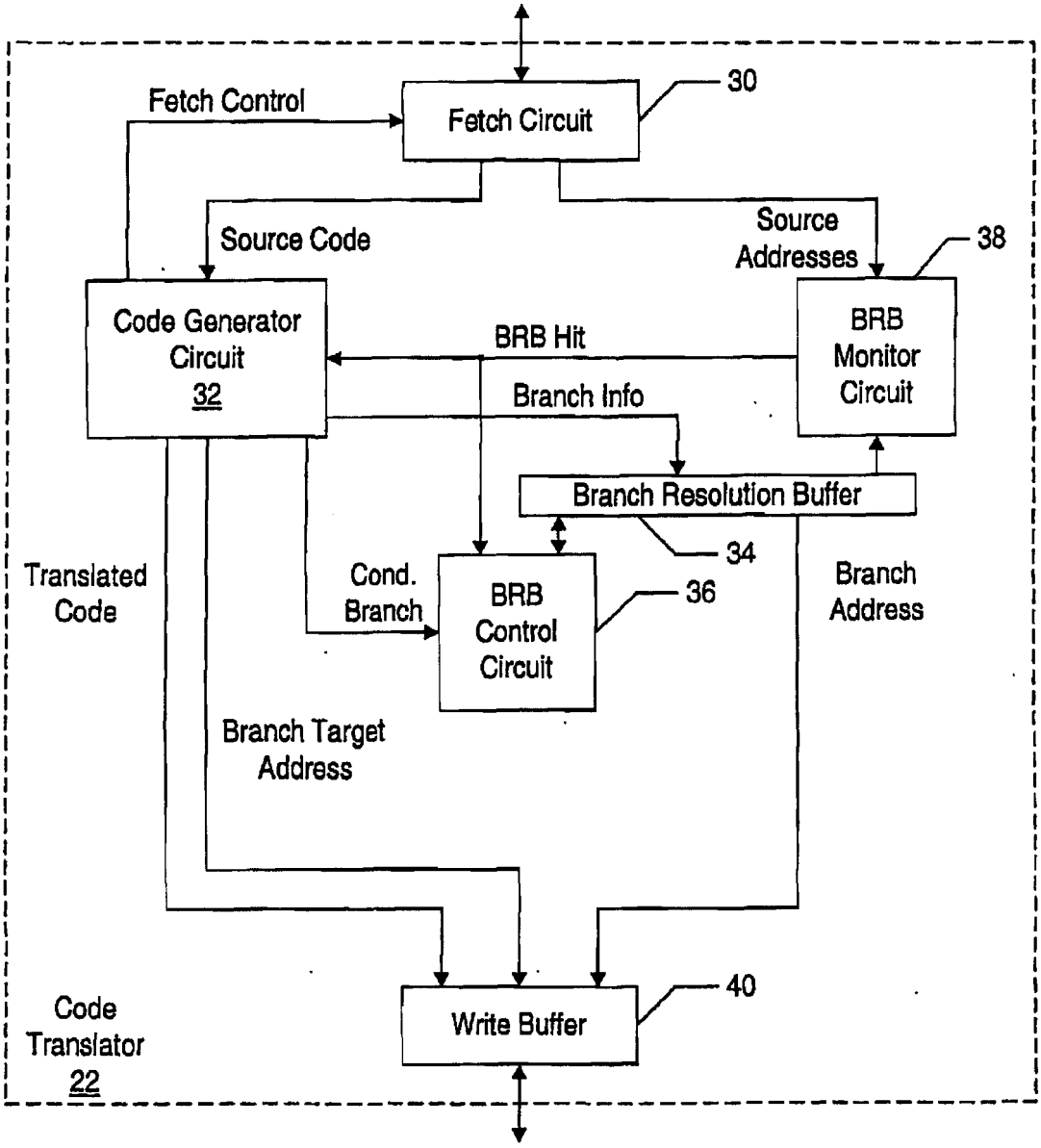

Fetch Control
Fetch Circuit
30
Source Code
Source Addresses
38
Code Generator Circuit 32
BRB Hit
BRB Monitor Circuit
Branch Info
Branch Resolution Buffer
34
Translated Code
Cond. Branch
BRB Control Circuit
36
Branch Address
Branch Target Address
40
Write Buffer
Code Translator 22

Fig. 2

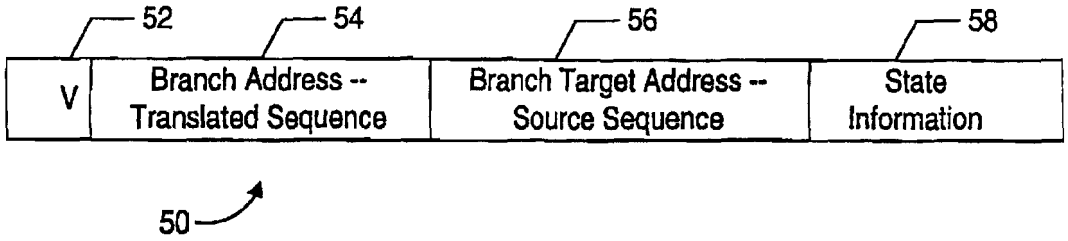

52
54
56
58
V
Branch Address -- Translated Sequence
Branch Target Address -- Source Sequence
State Information
50

Fig. 3

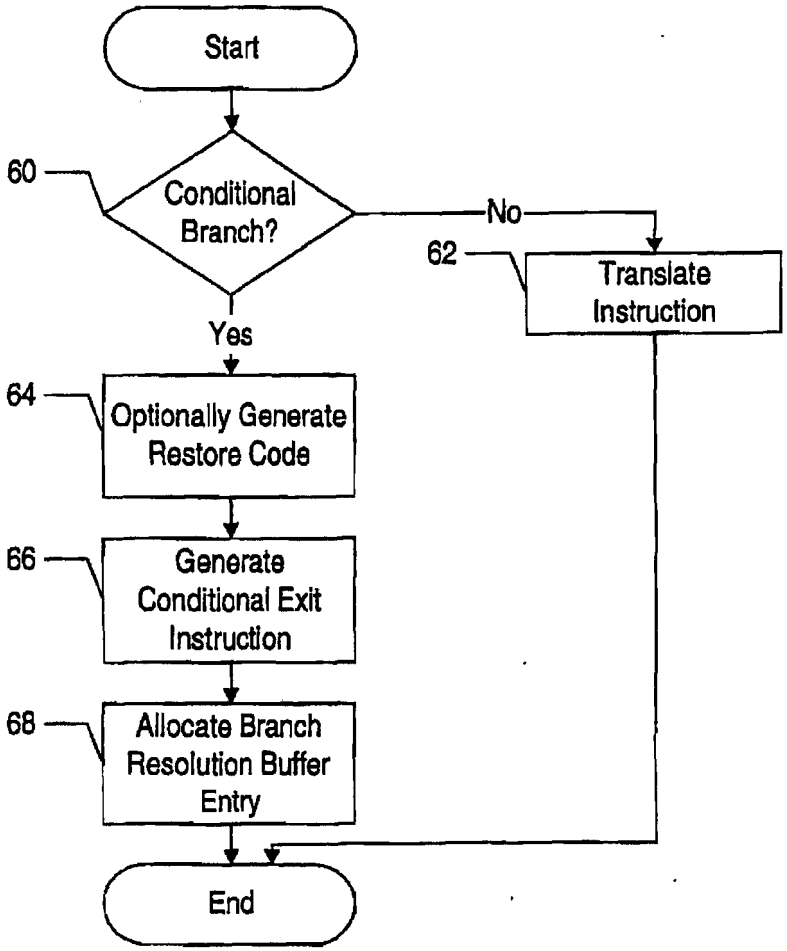

Start
60
Conditional Branch?
No
62
Translate Instruction
Yes
64
Optionally Generate Restore Code
66
Generate Conditional Exit Instruction
68
Allocate Branch Resolution Buffer Entry
End

Fig. 4

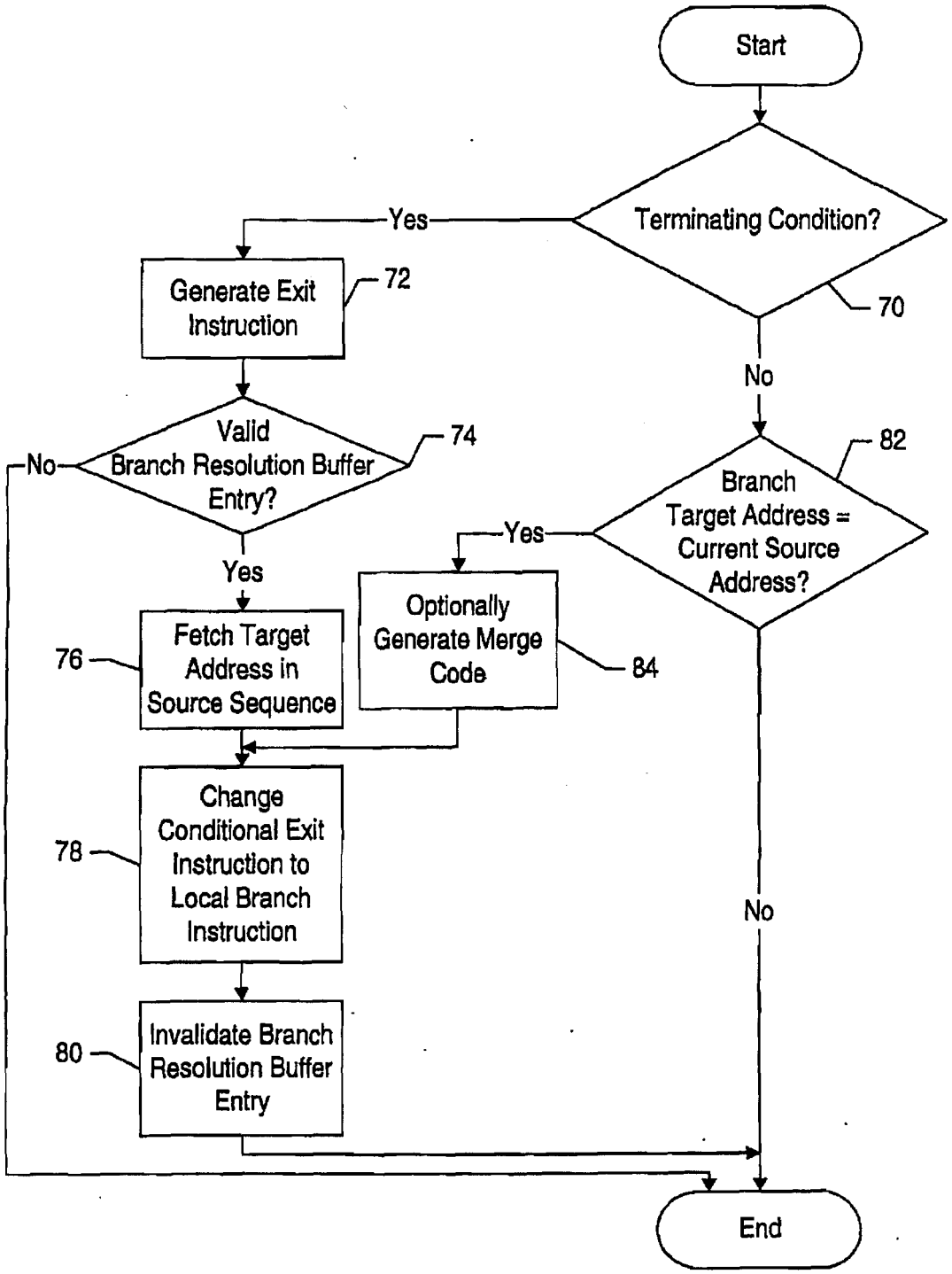

Start
Terminating Condition?
70
Yes
No
Generate Exit Instruction
72
Valid Branch Resolution Buffer Entry?
74
No
Yes
Fetch Target Address in Source Sequence
76
Change Conditional Exit Instruction to Local Branch Instruction
78
Invalidate Branch Resolution Buffer Entry
80
Branch Target Address = Current Source Address?
82
Yes
No
Optionally Generate Merge Code
84
End

Fig. 5

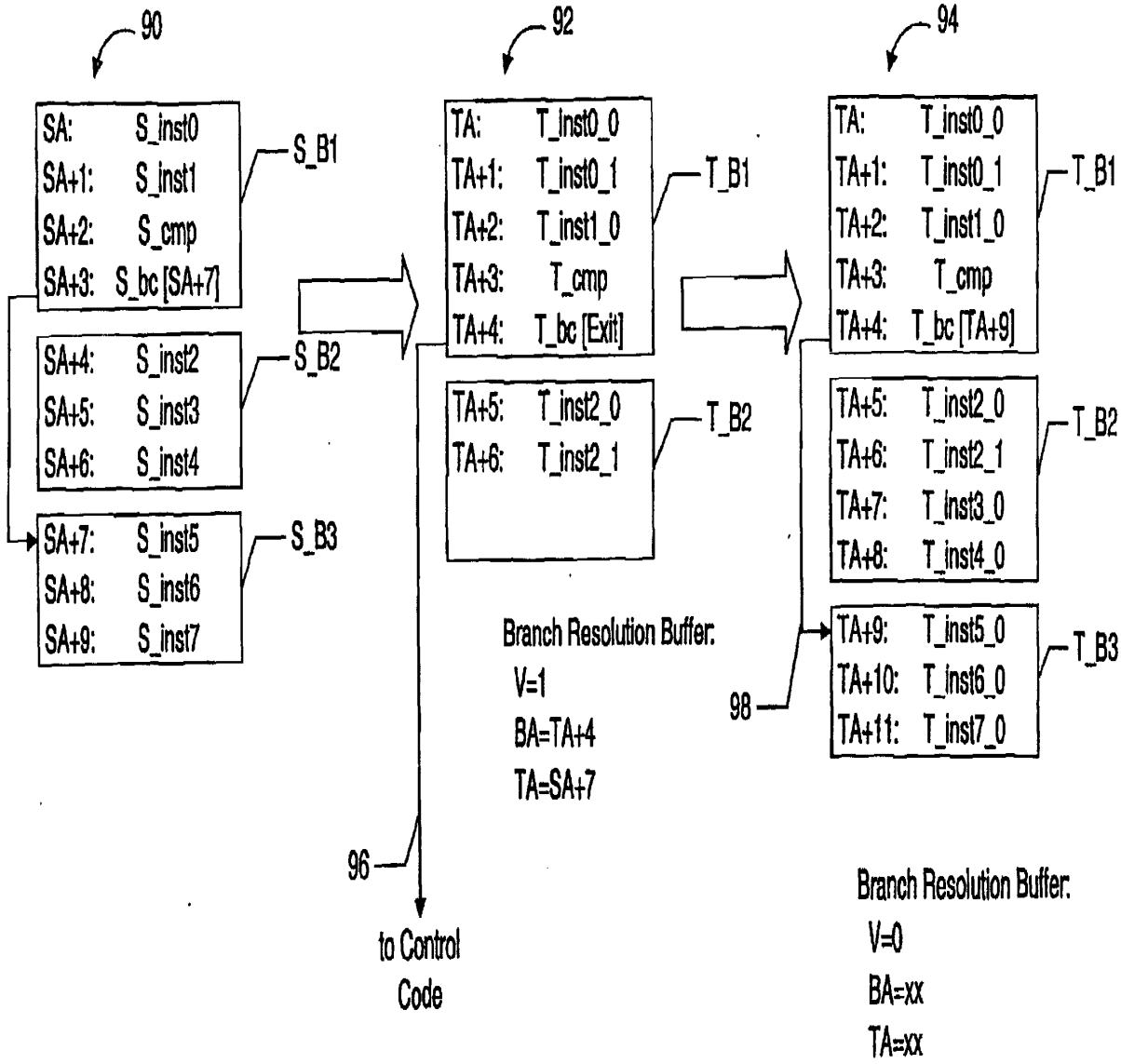

90
92
94
SA: S_inst0
SA+1: S_inst1
SA+2: S_cmp
SA+3: S_bc [SA+7]
S_B1
SA+4: S_inst2
SA+5: S_inst3
SA+6: S_inst4
S_B2
SA+7: S_inst5
SA+8: S_inst6
SA+9: S_inst7
S_B3
TA: T_inst0_0
TA+1: T_inst0_1
TA+2: T_inst1_0
TA+3: T_cmp
TA+4: T_bc [Exit]
T_B1
TA+5: T_inst2_0
TA+6: T_inst2_1
T_B2
Branch Resolution Buffer:
V=1
BA=TA+4
TA=SA+7
96
to Control Code
TA: T_inst0_0
TA+1: T_inst0_1
TA+2: T_inst1_0
TA+3: T_cmp
TA+4: T_bc [TA+9]
T_B1
TA+5: T_inst2_0
TA+6: T_inst2_1
TA+7: T_inst3_0
TA+8: T_inst4_0
T_B2
98
TA+9: T_inst5_0
TA+10: T_inst6_0
TA+11: T_inst7_0
T_B3
Branch Resolution Buffer:
V=0
BA=xx
TA=xx

Fig. 6

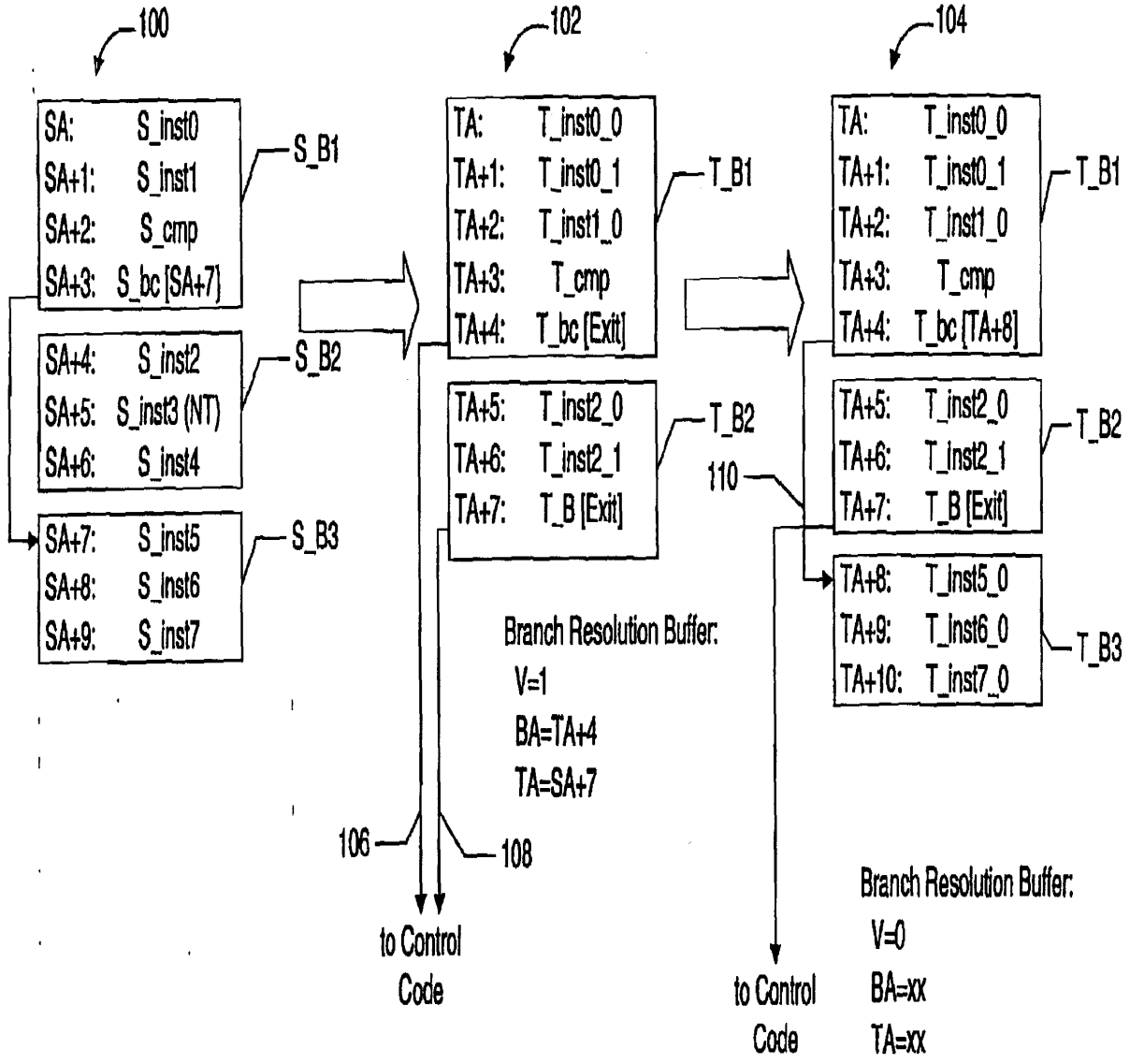

100
102
104
SA: S_inst0
SA+1: S_inst1
SA+2: S_cmp
SA+3: S_bc [SA+7]
S_B1
SA+4: S_inst2
SA+5: S_inst3 (NT)
SA+6: S_inst4
S_B2
SA+7: S_inst5
SA+8: S_inst6
SA+9: S_inst7
S_B3
TA: T_inst0_0
TA+1: T_inst0_1
TA+2: T_inst1_0
TA+3: T_cmp
TA+4: T_bc [Exit]
T_B1
TA+5: T_inst2_0
TA+6: T_inst2_1
TA+7: T_B [Exit]
T_B2
TA: T_inst0_0
TA+1: T_inst0_1
TA+2: T_inst1_0
TA+3: T_cmp
TA+4: T_bc [TA+8]
T_B1
TA+5: T_inst2_0
TA+6: T_inst2_1
TA+7: T_B [Exit]
T_B2
TA+8: T_inst5_0
TA+9: T_inst6_0
TA+10: T_inst7_0
T_B3
110
Branch Resolution Buffer:
V=1
BA=TA+4
TA=SA+7
106
108
to Control Code
to Control Code
Branch Resolution Buffer:
V=0
BA=xx
TA=xx

Fig. 7

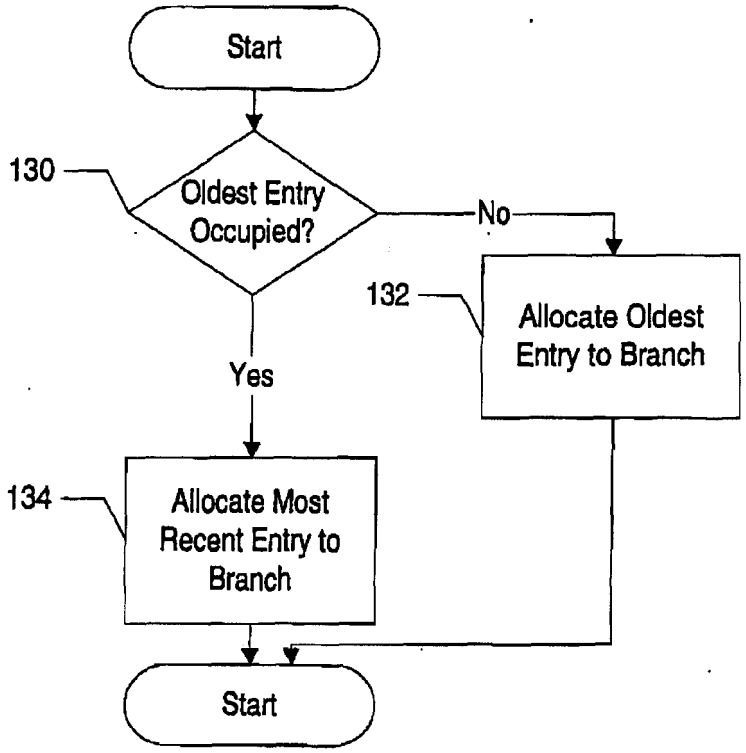

Start
130
Oldest Entry Occupied?
No
132
Allocate Oldest Entry to Branch
Yes
134
Allocate Most Recent Entry to Branch
Start

Fig. 8

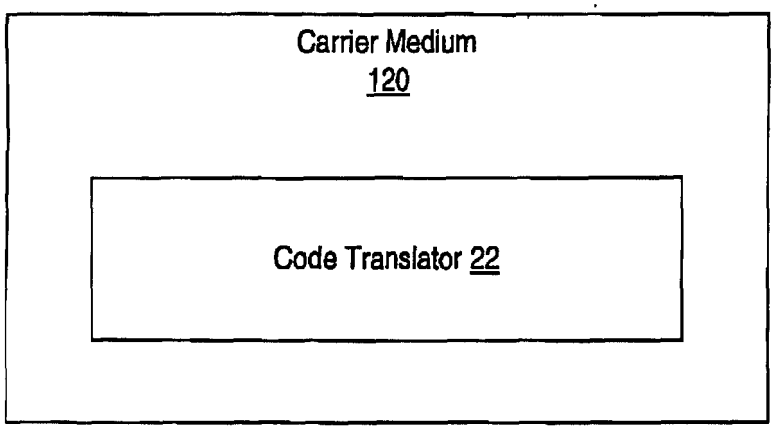


Fig. 9